Banyan Snap

Audit

Presented by:

OtterSec contact@osec.io

Robert Chen r@osec.io

Bruno Halltari bruno@osec.io

Caue Obici caue@osec.io

Contents

Appendices

© 2023 Otter Audits LLC. All Rights Reserved.

01 | Executive Summary

Overview

Banyan engaged OtterSec to perform an assessment of the Banyan Snap program. This assessment was conducted between September 11th, and September 12th, 2023. For more information on our auditing methodology, see Appendix B.

Key Findings

Over the course of this audit engagement, we produced 3 findings in total.

In particular, we reported issues related to Snap Dialog, which does not handle newlines properly, thereby facilitating message spoofing and phishing attacks (OS-BAN-ADV-00). Additionally, we identified a UI spoofing vulnerability that can be exploited when the apps are connecting, which may trick a user into allowing unsafe methods (OS-BAN-ADV-00).

We also made recommendations around the use of Spread syntax in order to avoid the overwrite of sensible attributes during future development (OS-BAN-SUG-00).

© 2023 Otter Audits LLC. All Rights Reserved.

02 | Scope

The source code was delivered to us in a git repository at github.com/here-wallet/near-snap. This audit was performed against commit 58a16b9.

A brief description of the programs is as follows:

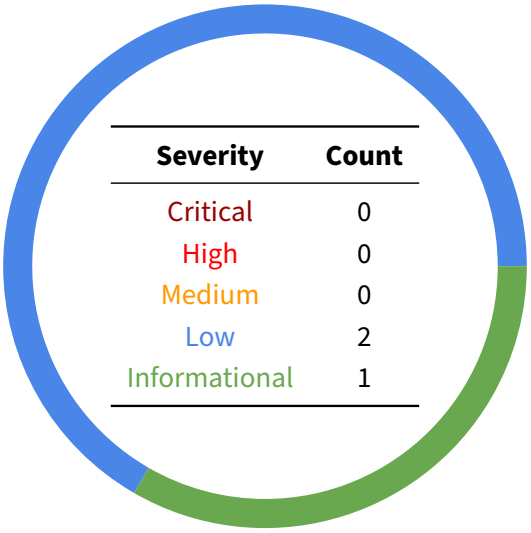
Name	Description
banyan-snap	Banyan is a Metamask Snap designed to sign transactions and provides an interface for viewing actions signed by the user inside the Near ecosystem.

© 2023 Otter Audits LLC. All Rights Reserved.

03 | Findings

Overall, we reported 3 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings do not have an immediate impact but will aid in mitigating future vulnerabilities.



Severity	Count
Critical	0
High	0
Medium	0
Low	2
Informational	1

© 2023 Otter Audits LLC. All Rights Reserved.

04 | Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

ID	Severity	Status	Description
OS-BAN-ADV-00	Low	Resolved	Snap dialogs can be spoofed via newlines injection
OS-BAN-ADV-01	Low	Resolved	Incorrect check against `params.methods` allows UI spoofing

© 2023 Otter Audits LLC. All Rights Reserved.

OS-BAN-ADV-00 [low] | Missing Newline Handling

Description

The Snap does not handle newlines properly inside Snap Dialog, which may allow for easier message spoofing and phishing attacks by integrating arbitrary text into the rest of the message. This may deceive users into taking certain actions on behalf of the Snap component.

The function handles messages passed to dialogs. However, the arguments are not correctly sanitized:

locales.ts TS

```ts
export const t = (key: string, ...args: any[]) => {
  const path = key.split('.');
  const txt = path.reduce(
    (root: any, k) => (typeof root === 'object' ? root[k] : null),
    locales,
  );

  if (!txt) {
    return key;
  }

  return txt.replaceAll(
    /\$\{(\d+)\}/gu,
    (_: any, i: number) => args[Number(i)],
  );
};
```

Proof of Concept

To practically demonstrate this vulnerability, we have created a proof of concept that leverages newlines to override the prompt message that may be used for phishing attacks:

JS

```js
snapId = "local:http://localhost:3000"

await window.ethereum.request({
      method: 'wallet_invokeSnap',
      params: {
        snapId: snapId,
        request: {method: 'near_connect', params: {network: "testnet", contractId:
            ↪  "\n\r\n\r\n\Go to evil.com in order to receive money"} },
      },
});e
```

© 2023 Otter Audits LLC. All Rights Reserved.

Remediation

Strip newlines from user input before creating dialogs. This can be done by sanitizing all `args` in the `t` function.

Patch

Fixed in 075cc27 by validating input data and removing newlines from strings.

```ts
STRING: (value: string) => {
  inputAssert(value, string());
  return value
    .replace(/[\r\n]/gmu, '')
    .replace(/\s+/gu, ' ')
    .trim();
},
```

© 2023 Otter Audits LLC. All Rights Reserved.

OS-BAN-ADV-01 [low] | UI Spoofing While Connecting Apps

Description

While connecting with an app using `near_connect` RPC method, the dApp can specify which methods need to be allowed via `request.params.methods`. If there are no specified methods, all methods will be allowed, and the ”allow all methods” message will appear in the confirmation dialog:

permissions.ts TS

```ts
if (params.contractId) {
const allowMethodsText = params.methods
? t('connectApp.allowMethods', params.contractId)
: t('connectApp.allowAllMethods', params.contractId)
```

However, this part of the code is mishandling empty arrays that also allow all methods, showing the `connectApp.allowMethods` message instead. This behavior may mislead the user into accepting unsafe permissions.

Remediation

Consider checking the `params.methods` against empty arrays and objects:

TS

```ts
const allMethods = params.methods ?
    params.methods.length == 0
    : true
```

Patch

Fixed in b28860f by checking `params.methods` against empty arrays and objects.

TS

```ts
    const isAllMethods = params.methods ? params.methods.length === 0 : true;
    const allowMethodsText = isAllMethods
      ? t('connectApp.allowAllMethods', params.contractId)
      : t('connectApp.allowMethods', params.contractId);
```

© 2023 Otter Audits LLC. All Rights Reserved.

05 | General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent anti-patterns and may result in security issues in the future.

ID	Description
OS-BAN-SUG-00	Spread syntax could allow critical attribute overwrite

© 2023 Otter Audits LLC. All Rights Reserved.

OS-BAN-SUG-00 | Incorrect Use Of Spread Syntax

Description

Spread syntax when creating an object (`{a: 1, ...obj}`) can overwrite previously defined attributes (a in this case). Thus, if an attacker can manipulate the spread object, they can overwrite potential critical attributes, as seen here:

index.ts TS

```ts
case Methods.ConnectApp:
      assert(request.params, connectWalletSchema);
      return await connectApp({ origin, snap, ...request.params });
```

In particular, if `request.params` had a `origin` attribute, it would overwrite the original one. However, the `assert` function does not allow it.

Remediation

Consider changing the spread syntax to the beginning of the object in future development when dealing with critical attributes for security.

Patch

Fixed in 219a220 by refactoring how arguments are passed.

© 2023 Otter Audits LLC. All Rights Reserved.

A | Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings may be found in the General Findings section.

Critical	Vulnerabilities that immediately result in a loss of user funds with minimal preconditions. Examples: • Misconfigured authority or access control validation. • Improperly designed economic incentives leading to loss of funds.
High	Vulnerabilities that may result in a loss of user funds but are potentially difficult to exploit. Examples: • Loss of funds requiring specific victim interactions. • Exploitation involving high capital requirement with respect to payout.
Medium	Vulnerabilities that may result in denial of service scenarios or degraded usability. Examples: • Computational limit exhaustion through malicious input. • Forced exceptions in the normal user flow.
Low	Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk. Examples: • Oracle manipulation with large capital requirements and multiple transactions.
Informational	Best practices to mitigate future security risks. These are classified as general findings. Examples: • Explicit assertion of critical internal invariants. • Improved input validation.

© 2023 Otter Audits LLC. All Rights Reserved.

B | Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the program's implementation requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of thumb, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to comprehensively understand the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that the other missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.

© 2023 Otter Audits LLC. All Rights Reserved.